SILOON

Scripting Interface Languages for Object-Oriented Numerics

SILOON (Scripting Interface Languages for Object-Oriented Numerics) gives users the ability to rapidly prototype their scientific codes in a simple yet elegant fashion using the popular scripting languages Python and Perl. While programming in these flexible and dynamic languages, SILOON users maintain the capability of accessing the full power and complexity of powerful FORTRAN and C++ libraries executed on high-performance parallel computers.

Overview of How SILOON Works

The SILOON toolkit parses source code from existing object-oriented numerical class libraries and extracts information regarding the interfaces to functions and class methods. This information is then used to generate glue and skeleton code, which when compiled, provides the run-time support for linking user scripts with back-end computational engines.

The parsing stage of SILOON is accomplished using Program Database Toolkit (PDT). PDT analyzes application source code and then makes this information available through a callable interface.

Because SILOON was designed within a client/server model, users can easily develop distributed applications. For example, a Perl scripting client may connect to an application running on a remote computational server, query the application about its current state, and then pipe data to a visualization tool. After examining the visual information, a user may wish to adjust run-time parameters in the application before continuing the computation.

Because SILOON (via PDT) uses a commercial ANSI-compliant parser (from the Edison Design Group), one of its unique features is its ability to handle many of the complexities of C++ correctly. These include

- templated classes and functions,
- virtual and static member functions,
- constructors and destructors,
- overloaded operators and functions,
- default function arguments,
- references
- enumerations,
- typedefs, and
- the Standard Template Library (STL).

In addition, because Python and Perl provide automatic garbage collection, SILOON adds this feature to the capabilities of C++.

Early versions of SILOON were targeted for use with the POOMA (Parallel Object-Oriented Methods and Applications) framework, the Visualization Interconnection Framework (VIF), and PAWS (Parallel Application WorkSpace).

How to Use SILOON

There are five essential steps to using SILOON. The following example assumes that there is a C++ source file, save_the_world.cpp, which includes the necessary header files defining the public interface exposed by SILOON.

1. Create and initialize the SILOON save_the_world project directories for use with Python.

```
siloon-init save_the_world --python
```

2. Parse user code and create the program interface database.

```
siloon-parse save_the_world.cpp
```

3. Generate scripting interfaces.

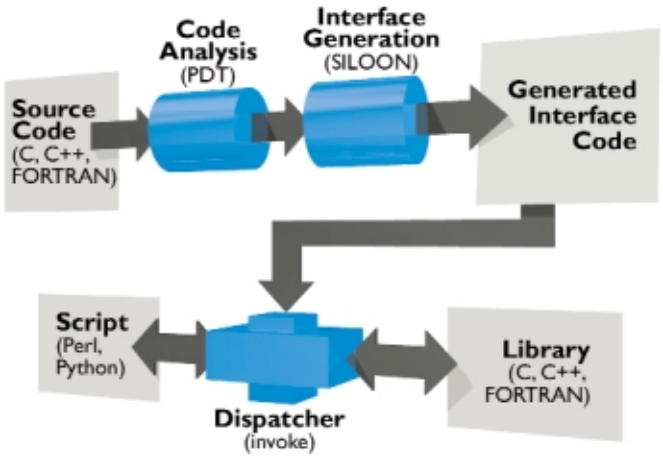
```
siloon-gen save_the_world.pdb
```

4. Compile and link.

```
make
```

5. Run interactively.

```
python
```



Code Examples

A series of examples are provided. In each example, the C++ source code is provided along with the Perl and Python calls.

Function Call

C++ function definition:

```
float temperature(int i, int j) { return 3.0; }
```

Python call:

```
T = temperature(0, 0)
```

Perl call:

```
$T = temperature(0, 0);
```

Member Function Invocation

C++ class definition:

```
class IntegerClass {
  private: int value;
  public: IntegerClass(int i) { value = i; }
  public: int getValue()      { return value; }
};
```

Python object creation and member function call:

```
iClass = IntegerClass(2)
value = iClass.getValue()
```

Perl object creation and member function call:

```
$iClass = IntegerClass->new(2);
$value = $iClass->getValue();
```

Templated Class

C++ class definition:

```
template <class T> class TClass {
  private: T value;
  public: TClass(T i)    { value = i; }
  public: T getValue()   { return value; }
};
```

Python object creation and member function call:

```
  tClass = TClass_float_(1.0)
  value = tClass.getValue()
```

Perl object creation and member function call:

```
  $tClass = TClass_float_->new(1.0);
  $value = $tClass->getValue();
```

Name Mangling

The example above shows the instantiation of a templated object of type TClass<float>. Since neither Python nor Perl support angle bracket notation ("<>"), a name mangling scheme has been chosen to handle these cases. The default mangling of TClass<float> is TClass_float_ as seen above. As name mangling can often be ugly and obtrusive, default names can be overridden via a file which provides a mapping between mangled names and function prototypes.

Suppose a user wanted to use the name TClassFloat for the TClass<float> constructor, rather than TClass_float_. One only need edit the prototypes file (initially generated automatically) and replace the default name "TClass_float_" in the first column with "TClassFloat" as shown below:

```
"TClassFloat" "TClass<float> &
               TClass<float>::TClass<float>(float)"
```

Names for overloaded functions may be specified in a similar manner.

More information about SILOON...
contact: Craig Rasmussen
e-mail: rasmussn@lanl.gov
web: www.acl.lanl.gov/siloon/

Get SILOON and other Advanced Computing Laboratory Software...
web: www.acl.lanl.gov/software/
cd: 1999 Advanced Computing Laboratory Software

This work supported by the US Department of Energy.

Los Alamos
NATIONAL LABORATORY

advanced computing laboratory

LALP-99-200 November 1999